A Quadratic Edge-Finding Filtering Algorithm for Cumulative Resource Constraints

Roger Kameugne[1,2], Laure Pauline Fotso[3], Joseph Scott[4], and Youcheu Ngo-Kateu[3]

[1] University of Maroua, Higher Teachers' Training College, Dept. of Mathematics, P.O. Box 55 Maroua-Cameroon
[2] University of Yaoundé I, Faculty of Sciences, Dept. of Mathematics, P.O. Box 812, Yaoundé, Cameroon
rkameugne@yahoo.fr, rkameugne@gmail.com
[3] University of Yaoundé I, Faculty of Sciences, Dept. of Computer Sciences, P.O. Box 812, Yaoundé, Cameroon
lpfotso@ballstate.bsu.edu, mireille_youcheu@yahoo.fr
[4] Uppsala University, Dept. of Information Technology, Computing Science Division, Box 337, SE-751 05 Uppsala Sweden
joseph.scott@it.uu.se

Abstract. The cumulative scheduling constraint, which enforces the sharing of a finite resource by several tasks, is widely used in constraint-based scheduling applications. Propagation of the cumulative constraint can be performed by several different filtering algorithms, often used in combination. One of the most important and successful of these filtering algorithms is edge-finding. Recent work by Vilím has resulted in a $\mathcal{O}(kn \log n)$ algorithm for cumulative edge-finding, where n is the number of tasks and k is the number of distinct capacity requirements. In this paper, we present a sound $\mathcal{O}(n^2)$ cumulative edge-finder. This algorithm reaches the same fixpoint as previous edge-finding algorithms, although it may take additional iterations to do so. While the complexity of this new algorithm does not strictly dominate Vilím's for small k, experimental results on benchmarks from the Project Scheduling Problem Library suggest that it typically has a substantially reduced runtime. Furthermore, the results demonstrate that in practice the new algorithm rarely requires more propagations than previous edge-finders.

1 Introduction

Edge-finding is a filtering technique commonly used in solving resource-constrained project scheduling problems (RCPSP). An RCPSP consists of a set of resources of finite capacities, a set of tasks of given processing times, an acyclic network of precedence constraints between tasks, and a horizon (a deadline for all tasks). Each task requires a fixed amount of each resource over its execution time. The problem is to find a start time assignment for every task satisfying the precedence and resource capacity constraints, with a makespan (i.e., the time at which all tasks are completed) equal at most to the horizon. Edge-finding reduces

J. Lee (Ed.): CP 2011, LNCS 6876, pp. 478–492, 2011.

the range of possible start times by deducing new ordering relations between the tasks: for a task i, an edge-finder searches for a set of tasks Ω that *must* end before the end (or alternately, start before the start) of i. Based on this newly detected precedence, the earliest start time (or latest completion time) of i is updated. Note that we consider here only non-preemptive scheduling problems; that is, once started a task executes without interruption until it is completed. For disjunctive scheduling (i.e., scheduling on a resource of capacity $C = 1$) on a set of n tasks, there are well-known $\mathcal{O}(n \log n)$ edge-finding algorithms [3], [11]. In cumulative scheduling, where tasks may have different capacity requirements, edge-finding is more challenging. Early work by Nuijten [9] and Baptiste [2] resulted in cumulative edge-finding algorithms of complexity $\mathcal{O}(n^2k)$ (where k is the number of distinct capacity requirements among the tasks) and $\mathcal{O}(n^2)$, respectively. Mercier and Van Hentenryck [8] demonstrated that both of these algorithms were incomplete, and provided a correct $\mathcal{O}(n^2k)$ algorithm. More recently, Vilím [13] gave a $\mathcal{O}(kn \log n)$ algorithm, using an extension of the Θ-tree data structure which had previously been used [11] to improve the complexity of disjunctive edge-finding.

In this paper, we present a new cumulative edge-finding algorithm with a complexity of $\mathcal{O}(n^2)$. This algorithm uses the maximum density and minimum slack of sets of tasks to quickly locate the set Θ that provides the strongest update to the bounds of i.

The paper is organized as follows. Section 2 defines the cumulative scheduling problem, and the notations used in the paper. Section 3 gives a formal definition of the edge finding rules; Section 4 provides dominance properties of these rules. Section 5 presents the new edge-finding algorithm and a proof of its soundness. Section 6 discusses the overall complexity of the algorithm. Section 7 reports experimental results.

2 Cumulative Scheduling Problem

The cumulative scheduling problem (CuSP) is a sub-problem of the RCPSP, where precedence constraints are relaxed and a single resource is considered at a time; both problems are NP-complete [1]. In a CuSP, there is a finite set of tasks or activities with fixed processing times and resource requirements. Each task has a defined earliest start and latest completion time. The problem consists of deciding when to execute each task so that time and resource constraints are satisfied. Tasks are assumed to be processed without interruption. Formally, this problem is defined as follows:

Definition 1 (Cumulative Scheduling Problem). *A Cumulative Scheduling Problem (CuSP) is defined by a set T of tasks to be performed on a resource of capacity C. Each task $i \in T$ must be executed without interruption over p_i units of time between an earliest start time r_i (release date) and a latest end time d_i (deadline). Moreover, each task requires a constant amount of resource c_i. It is assumed that all data are integer. A solution of a CuSP is a schedule that assigns*

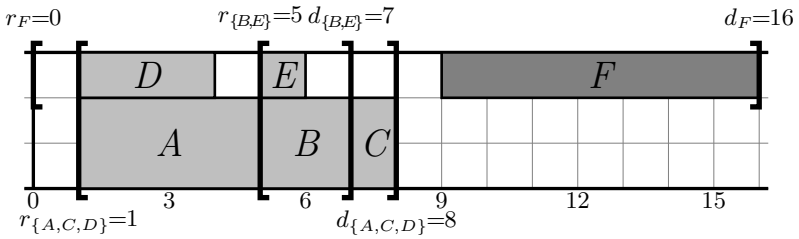


Fig. 1. A scheduling problem of 6 tasks sharing a resource of capacity $C = 3$

a starting date s_i to each task i such that:

$$\forall i \in T\colon r_i \leq s_i \leq s_i + p_i \leq d_i \tag{1}$$

$$\forall \tau\colon \sum_{i \in T,\ s_i \leq \tau < s_i + p_i} c_i \leq C \tag{2}$$

The inequalities in (1) ensure that each task is assigned a feasible start and end time, while (2) enforces the resource constraint. An example of a CuSP is given in Fig. 1.

We define the energy of a task i as $e_i = c_i \cdot p_i$. This notation, along with that of earliest start and latest completion time, may be extended to non-empty sets of tasks as follows:

$$r_\Omega = \min_{j \in \Omega} r_j, \quad d_\Omega = \max_{j \in \Omega} d_j, \quad e_\Omega = \sum_{j \in \Omega} e_j \tag{3}$$

By convention, if Ω is the empty set, $r_\Omega = +\infty$, $d_\Omega = -\infty$, and $e_\Omega = 0$. Throughout the paper, we assume that for any task $i \in T$, $r_i + p_i \leq d_i$ and $c_i \leq C$, otherwise the problem has no solution. We let $n = |T|$ denote the number of tasks, and $k = |\{c_i, i \in T\}|$ denote the number of distinct capacity requirements.

Clearly, if there exists a set of tasks $\Omega \subseteq T$ which cannot be scheduled in the window from r_Ω to d_Ω without exceeding the capacity, then the CuSP has no feasible solution. *Overload checking* algorithms typically enforce the following relaxation of this feasibility condition, which may be computed in $\mathcal{O}(n \log n)$ time [12], [14]:

Definition 2 (E-Feasibility). *[8] A problem is E-feasible if* $\forall \Omega \subseteq T,\ \Omega \neq \emptyset$

$$C\,(d_\Omega - r_\Omega) \geq e_\Omega\ . \tag{4}$$

It is obvious that a CuSP that fails the E-feasibility condition cannot have a feasible solution. In the rest of the paper, we only consider E-feasible CuSPs.

3 The Edge-Finding Rule

The main idea of edge-finding is to discover a set of tasks $\Omega \subset T$ and a task $i \notin \Omega$ such that, in any solution, all the tasks in Ω end before the end of i;

following [13], we denote this relationship $\Omega \lessdot i$. Once an appropriate Ω and i have been located, the earliest start time of i can be adjusted using the following rule:

$$\Omega \lessdot i \implies r_i \geq r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil \tag{5}$$

for all $\Theta \subseteq \Omega$ such that $\operatorname{rest}(\Theta, c_i) > 0$, where

$$\operatorname{rest}(\Theta, c_i) = \begin{cases} e_\Theta - (C - c_i)\,(d_\Theta - r_\Theta) & \text{if } \Theta \neq \emptyset \\ 0 & \text{otherwise} \end{cases} . \tag{6}$$

The condition $\operatorname{rest}(\Theta, c_i) > 0$ states that the total energy e_Ω that must be scheduled in the window $[r_\Omega, d_\Omega)$ is strictly larger than the energy that could be scheduled without making any start time of i in that window infeasible. The proof of these results can be found in [2], [9].

It remains to define what tasks and sets of tasks satisfy the condition $\Omega \lessdot i$. Proposition 1 provides conditions under which all tasks of a set Ω of an E-feasible CuSP end before the end of a task i.

Proposition 1. *Let Ω be a set of tasks and let $i \notin \Omega$ be a task of an E-feasible CuSP.*

$$e_{\Omega \cup \{i\}} > C\left(d_\Omega - r_{\Omega \cup \{i\}}\right) \Rightarrow \Omega \lessdot i \ , \tag{EF}$$

$$r_i + p_i \geq d_\Omega \quad \Rightarrow \quad \Omega \lessdot i \ . \tag{EF1}$$

Proof. (EF) is the traditional edge-finding rule; proof can be found in [9,2]. The addition of (EF1), proposed in [13], strengthens the edge-finding rule; the proof follows trivially from the fact that $r_i + p_i \geq d_\Omega$ implies that task i ends before all tasks in the set Ω. □

In the example shown in Fig. 1, the rule (EF) correctly detects $\Omega \lessdot F$ for $\Omega = \{A, B, C, D, E\}$. Using the set $\Theta = \Omega$ in formula (5), shows that the release date of F may be updated to 5; however, allowing $\Theta = \{B, E\}$ instead yields an updated bound of 6. A value of $\Omega = \{B, E\}$ would not meet the edge-finding condition in (EF); the set $\{A, B, C, D, E\}$ is needed to detect the precedence condition.

Combining (EF) and (EF1) with (5) gives us a formal definition of an edge-finding algorithm:

Definition 3 (Specification of a complete edge-finding algorithm). *An edge-finding algorithm receives as input an E-feasible CuSP, and produces as output a vector of updated lower bounds for the release times of the tasks $\langle LB_1, \ldots, LB_n \rangle$, where:*

$$LB_i = \max\left(r_i, \max_{\substack{\Omega \subseteq T \\ i \notin \Omega \\ \alpha(\Omega, i)}} \max_{\substack{\Theta \subseteq \Omega \\ \operatorname{rest}(\Theta, c_i) > 0}} r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil\right) \tag{7}$$

with

$$\alpha\left(\Omega, i\right) \stackrel{def}{=} \left(C\left(d_\Omega - r_{\Omega\cup\{i\}}\right) < e_{\Omega\cup\{i\}}\right) \vee \left(r_i + p_i \geq d_\Omega\right) \tag{8}$$

and

$$\text{rest}(\Theta, c_i) = \begin{cases} e_\Theta - (C - c_i)\left(d_\Theta - r_\Theta\right) & \text{if } \Theta \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

4 Dominance Properties of the Rules

Clearly an edge-finder cannot efficiently consider all sets $\Theta \subseteq \Omega \subset T$ to update a task i. In order to reduce the number of sets which must be considered, we first consider the following definition:

Definition 4 (Task Intervals). *(After [4]) Let $L, U \in T$. The task intervals $\Omega_{L,U}$ is the set of tasks*

$$\Omega_{L,U} = \{j \in T \mid r_L \leq r_j \wedge d_j \leq d_U\} \quad . \tag{10}$$

It is demonstrated in [8] that an edge-finder that only considers sets $\Omega \subseteq T$ and $\Theta \subseteq \Omega$ which are also task intervals can be complete. Furthermore, we can reduce the number of intervals that must be checked according to the following propositions:

Proposition 2. *[8] Let i be a task and Ω, Θ be two task sets of an E-feasible CuSP with $\Theta \subseteq \Omega$. If the edge-finding rule* (EF) *applied to task i with pair (Ω, Θ) allows to update the earliest start time of i then*

(i) there exists four tasks L, U, l, u such that $r_L \leq r_l < d_u \leq d_U < d_i \wedge r_L \leq r_i$
(ii) the edge-finding rule (EF) *applied to task i with the pair $(\Omega_{L,U}, \Omega_{l,u})$ allows at least the same update of the earliest start time of task i.*

Proposition 3. *Let i be a task and Ω, Θ be two task sets of an E-feasible CuSP with $\Theta \subseteq \Omega$. If the edge-finding rule (EF1) applied to task i with pair (Ω, Θ) allows to update the earliest start time of i then*

(i) there exists four tasks L, U, l, u such that $r_L \leq r_l < d_u \leq d_U < d_i$
(ii) the edge-finding rule (EF1) *applied to task i with the pair $(\Omega_{L,U}, \Theta_{l,u})$ allows at least the same update of the earliest start time of task i.*

Proof. The proof of Proposition 3 is similar to that of Proposition 2, given in [8].

5 A New Edge-Finding Algorithm

In this section, we present a quadratic edge-finding algorithm that reaches the same fix point as the well known edge-finding algorithm proposed by Vilím [13].

We start by considering the $\mathcal{O}(n^2)$ edge-finding algorithm proposed in [2] and [9], which finds the set Θ for the inner maximization of Definition 3 by locating the task intervals with the minimum *slack*, as given by the following definitions.

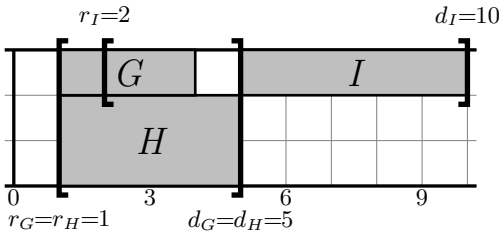


Fig. 2. Three tasks to be scheduled on a resource of capacity $C = 3$

Definition 5. *Let Ω be a task set of an E-feasible CuSP. The* slack *of the task set Ω, denoted SL_Ω, is given by: $SL_\Omega = C(d_\Omega - r_\Omega) - e_\Omega$.*

Definition 6. *Let i and U be two tasks of an E-feasible CuSP. $\tau(U, i)$, where $r_{\tau(U,i)} \leq r_i$, defines the task intervals with the* minimum slack*: for all $L \in T$ such that $r_L \leq r_i$,*

$$C(d_U - r_{\tau(U,i)}) - e_{\Omega_{\tau(U,i),U}} \leq C(d_U - r_L) - e_{\Omega_{L,U}} \ .$$

For a given task i, the algorithm detects $\Omega \ll i$ by computing $SL_\Omega < e_i$ for all $\Omega = \Omega_{\tau(U,i),U}$ such that $d_U < d_i$ and $r_{\tau(U,i)} \leq r_i$. Furthermore, if the interval $\Theta_{l,u}$ that yields the strongest update to r_i is such that $r_l \leq r_i$, then $\Theta_{l,u}$ will be the interval of minimum slack. This is the situation shown in Fig. 2. Rather than determine r_l, the new algorithm computes a potential update to r_i using rest(Ω, c_i). However, if the strongest updating interval $\Theta_{l,u}$ has $r_i < r_l$, then $\Theta_{l,u}$ need not be the interval of minimum slack. For this case, we introduce the notion of interval *density*.

Definition 7. *Let Θ be a task set of an E-feasible CuSP. The* density *of the task set Θ, denoted $Dens_\Theta$, is given by: $Dens_\Theta = \frac{e_\Theta}{d_\Theta - r_\Theta}$.*

Definition 8. *Let i, u be two tasks of an E-feasible CuSP. $\rho(u, i)$, where $r_i < r_{\rho(u,i)}$, defines the task intervals with the* maximum density*: for all task $l \in T$ such that $r_i < r_l$,*

$$\frac{e_{\Theta_{l,u}}}{d_u - r_l} \leq \frac{e_{\Theta_{\rho(u,i),u}}}{d_u - r_{\rho(u,i)}} \ .$$

If the strongest updating interval $\Theta_{l,u}$ has $r_i < r_l$, then $\Theta_{l,u}$ will be the interval of maximum density. Consequently, the new algorithm computes a second potential update for each r_i: for each task u such that $d_u \leq d_U$, the task intervals of maximum density $\Theta_{\rho(u,i),u}$ with $r_{\rho(u,i)} > r_i$ is computed according to Definition 8, and the strongest update of any of these intervals becomes the second potential update. The algorithm does not determine in advance whether $r_l \leq r_i$ or not; that is, whether minimum slack or maximum density is the correct method to locate the strongest update. For each i, both potential updates are computed, and the stronger update is applied.

Consider the scheduling problem shown in Fig. 1. With $\Omega = \{A, B, C, D, E\}$ and $i = F$, the rule (EF) detects the condition $\Omega \ll i$. However, the algorithm from [9] fails to adjust r_F, because for all $\Theta \subseteq \Omega$ we have $r_\Theta > r_F$. We can update r_F using a task intervals of maximum density, $\Theta_{\rho(u,i),u}$ where $d_u \leq d_\Omega$. For $u \in \{A, C, D\}$, the interval of maximum density is $\Theta_{A,D} = \{A, B, C, D, E\}$, which has a density of $18/7 \approx 2.6$. Using (5) with $\Theta_{A,D}$ shows that we can strengthen the release date of F to $r_F \geq 5$. For $u \in \{B, E\}$, however, the interval of maximum density is $\Theta_{B,E} = \{B, E\}$, which has a density of $5/2 = 2.5$. Using (5) with $\Theta_{B,E}$ yields a new release date for F of $r_F \geq 6$, which is in fact the strongest update we can make.

Algorithm 1 performs these computations for all $i \in T$ in $\mathcal{O}(n^2)$ time. The outer loop (line 1) iterates through the tasks $U \in T$ forming the possible upper bounds of the task intervals, selected in the order of non-decreasing deadlines. The first inner loop (line 1) selects the tasks $i \in T$ that comprise the possible lower bounds for the task intervals, in non-increasing order by release date. If $d_i \leq d_U$, then the energy and density of $\Omega_{i,U}$ are calculated; if the new density is higher than $\Omega_{\rho(U,i),U}$, $\rho(U, i)$ becomes i. If $d_i > d_U$, then instead the potential update $Dupd_i$ to the release date of i is calculated, based on the current $\rho(U, i)$. This potential update is stored only if it is greater than the previous potential update value calculated for this task using the maximum density. The second inner loop (line 1) selects i in non-decreasing order by release date. The energies stored in the previous loop are used to compute the slack of the current interval $\Omega_{i,U}$. If the slack is lower than that of $\Omega_{\tau(U,i),U}$, $\tau(U, i)$ becomes i. Any task with a deadline greater than d_U is checked to see if it meets either edge-finding criteria (EF) or (EF1); if it does, a new potential update $SLupd_i$ for the task's release date is calculated using $\tau(U, i)$. This potential update is stored only if it is greater than the previous potential update value calculated for this task using the minimum slack. At the next iteration of the outer loop, $\rho(U, i)$ and $\tau(U, i)$ are re-initialized.

Before showing that Algorithm 1 is correct, let us prove some properties of its inner loops.

Proposition 4. *For each task i, Algorithm 1 calculates a potential update $Dupd_i$ to r_i based on the task intervals of maximum density such that*

$$Dupd_i = \max_{U\,:\, d_U < d_i \wedge \text{rest}(\Theta_{\rho(U,i),U}, c_i) > 0} \left(r_{\rho(U,i)} + \left\lceil \text{rest}(\Theta_{\rho(U,i),U}, c_i) \cdot \frac{1}{c_i} \right\rceil \right) \quad . \quad (11)$$

Proof. Let $i \in T$ be any task. Each choice of $U \in T$ in the outer loop (line 1) starts with the values $r_\rho = -\infty$ and $maxEnergy = 0$. The inner loop at line 1 iterates through all tasks $i' \in T$ (T sorted in non-increasing order of release date). For any task $i' \in T$ such that $r_{i'} > r_i$, if $d_{i'} \leq d_U$, then $i' \in \Theta_{i',U}$, so $e_{i'}$ is added to $Energy$ (line 1). Hence $Energy = e_{\Theta_{i',U}}$ at each iteration. The test on line 1 ensures that r_ρ and $maxEnergy = e_{\Theta_{\rho,U}}$ are updated to reflect $\rho(U, i)$ for the current task intervals $\Theta_{i',U}$. Therefore, at the i^{th} iteration of the inner loop, if $d_i > d_U$ then line 1 computes $\text{rest}(i, U) = \text{rest}(\Theta_{\rho(U,i),U}, c_i)$, and the potential update value: $r_\rho + \lceil \text{rest}_{i,U} \cdot \frac{1}{c_i} \rceil$ if $\text{rest}(i, U) > 0$, and $-\infty$ otherwise. On line 1,

Algorithm 1. Edge-finding algorithm in $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space

Require: T is an array of tasks
Ensure: A lower bound LB'_i is computed for the release date of each task i

1 **for** $i \in T$ **do**
2 $\quad LB'_i := r_i, \quad Dupd_i := -\infty, \quad SLupd_i := -\infty;$
3 **for** $U \in T$ *by non-decreasing deadline* **do**
4 $\quad Energy := 0, \quad maxEnergy := 0, \quad r_\rho := -\infty;$
5 $\quad$ **for** $i \in T$ *by non-increasing release dates* **do**
6 $\quad\quad$ **if** $d_i \leq d_U$ **then**
7 $\quad\quad\quad Energy := Energy + e_i;$
8 $\quad\quad\quad$ **if** $\left(\frac{Energy}{d_U - r_i} > \frac{maxEnergy}{d_U - r_\rho}\right)$ **then**
9 $\quad\quad\quad\quad maxEnergy := Energy, \quad r_\rho := r_i\ ;$
10 $\quad\quad$ **else**
11 $\quad\quad\quad rest := maxEnergy - (C - c_i)(d_U - r_\rho);$
12 $\quad\quad\quad$ **if** $(rest > 0)$ **then**
13 $\quad\quad\quad\quad Dupd_i := \max(Dupd_i, \quad r_\rho + \lceil \frac{rest}{c_i} \rceil);$
14 $\quad\quad E_i := Energy\ ;$
15 $\quad minSL := +\infty,\ r_\tau := d_U\ ;$
16 $\quad$ **for** $i \in T$ *by non-decreasing release date* **do**
17 $\quad\quad$ **if** $(C(d_U - r_i) - E_i < minSL)$ **then**
18 $\quad\quad\quad r_\tau := r_i, \quad minSL := C(d_U - r_\tau) - E_i\ ;$
19 $\quad\quad$ **if** $(d_i > d_U)$ **then**
20 $\quad\quad\quad rest' := c_i(d_U - r_\tau) - minSL\ ;$
21 $\quad\quad\quad$ **if** $(r_\tau \leq d_U \wedge rest' > 0)$ **then**
22 $\quad\quad\quad\quad SLupd_i := max(SLupd_i, r_\tau + \lceil \frac{rest'}{c_i} \rceil);$
23 $\quad\quad\quad$ **if** $(r_i + p_i \geq d_U \vee minSL - e_i < 0)$ **then**
24 $\quad\quad\quad\quad LB'_i := \max(LB'_i, Dupd_i, SLupd_i);$
25 **for** $i \in T$ **do**
26 $\quad r_i := LB'_i;$

$Dupd_i$ is updated only if $r_\rho + \lceil \text{rest}(i, U) \cdot \frac{1}{c_i} \rceil$ is larger than the current value of $Dupd_i$; since the outer loop selects the task U in non-decreasing order by d_U, we have:

$$Dupd_i = \max_{U:\ d_U < d_i \wedge \text{rest}(i,U) > 0} r_\rho + \lceil \text{rest}(i, U) \cdot \frac{1}{c_i} \rceil\ . \tag{12}$$

Hence formula (11) holds and the proposition is correct. □

Proposition 5. *For each task i, Algorithm 1 calculates a potential update $SLupd_i$ to r_i based on the task intervals of minimum slack such that*

$$SLupd_i = \max_{U:\ d_U < d_i \wedge \text{rest}(\Omega_{\tau(U,i),U}, c_i) > 0} \left(r_{\tau(U,i)} + \left\lceil \text{rest}(\Omega_{\tau(U,i),U}, c_i) \cdot \frac{1}{c_i} \right\rceil \right)\ . \tag{13}$$

Proof. Let $i \in T$ be any task. Each choice of U in the outer loop (line 1) starts with the values $r_\tau = d_U$ and $minSL = +\infty$ (line 1). The inner loop at line 1

iterates through all tasks $i' \in T$ (T sorted in non-decreasing order of release dates). For every task $i' \in T$, $e_{\Omega_{i',U}}$ has already been computed in the first loop and stored as $E_{i'}$ (line 1); this is used to compute the slack of $\Omega_{i',U}$. If $SL_{\Omega_{i',U}} < minSL$, the values $r_\tau = r_{i'}$ and $minSL = C(d_U - r_{i'}) - e_{\Omega_{i',U}}$ are updated to reflect $\tau(U,i)$ for the current task intervals $\Omega_{i',U}$. At the i^{th} iteration, if $d_i > d_U$, then line 1 computes $rest'(i,U) = \text{rest}(\Omega_{\tau(U,i),U}, c_i)$, and the potential update value: $r_\tau + \lceil \text{rest}'(i,U) \cdot \frac{1}{c_i} \rceil$ if $r_\tau \leq d_U \wedge \text{rest}'(i,U) > 0$, and $-\infty$ otherwise. On line 1, $SLupd_i$ is updated only if $r_\tau + \lceil \text{rest}'(i,U) \cdot \frac{1}{c_i} \rceil$ is larger than the current value of $SLupd_i$. Since the outer loop selects the task U in non-decreasing order by d_U we have:

$$SLupd_i = \max_{U\,:\,d_U \leq d_i \wedge r_\tau \leq d_U \wedge \text{rest}'(i,U) > 0} r_\tau + \lceil \text{rest}'(i,U) \cdot \frac{1}{c_i} \rceil \ . \tag{14}$$

Hence formula (13) holds and the proposition is correct. □

We now provide a proof that the edge-finding condition (EF) can be checked using minimum slack.

Theorem 1. *For any task $i \in T$ and set of tasks $\Omega \subseteq T \setminus \{i\}$,*

$$e_{\Omega\cup\{i\}} > C(d_\Omega - r_{\Omega\cup\{i\}}) \vee r_i + p_i \geq d_\Omega \tag{15}$$

if and only if

$$e_i > C\left(d_U - r_{\tau(U,i)}\right) - e_{\Omega_{\tau(U,i),U}} \vee r_i + p_i \geq d_U \tag{16}$$

for some task $U \in T$ such that $d_U < d_i$, and $\tau(U,i)$ as specified in Definition 6.

Proof. Let $i \in T$ be any task.

It is obvious that (EF1) can be checked by $r_i + p_i \geq d_U$ for all tasks $U \in T$ with $d_i > d_\Omega$. In the rest of the proof, we focus on the rule (EF). We start by demonstrating that (15) implies (16). Assume there exists a subset $\Omega \subseteq T \setminus \{i\}$ such that $C(d_\Omega - r_{\Omega\cup\{i\}}) < e_\Omega + e_i$. By (EF), $\Omega \lessdot i$. By Proposition 2, there exists a task intervals $\Omega_{L,U} \lessdot i$, such that $d_i > d_U$ and $r_L \leq r_i$. By Definition 6 we have

$$C(d_U - r_{\tau(U,i)}) - e_{\Omega_{\tau(U,i),U}} \leq C(d_U - r_L) - e_{\Omega_{L,U}} \ . \tag{17}$$

Adding $-e_i$ to both sides of (17) and using the fact that

$$C(d_U - r_L) < e_{\Omega_{L,U}} + e_i \ , \tag{18}$$

it follows that

$$C\left(d_U - r_{\tau(U,i)}\right) < e_{\Omega_{\tau(U,i),U}} + e_i \ . \tag{19}$$

Now we show that (16) implies (15). Let $U \in T$ such that $d_U < d_i$, and $\tau(U,i) \in T$, be tasks that satisfy (16). By the definition of task intervals, $d_i > d_U$ implies $i \notin \Omega_{\tau(U,i),U}$. Since $r_{\tau(U,i)} \leq r_i$, we have

$$e_{\Omega_{\tau(U,i),U}} + e_i > C(d_U - r_{\tau(U,i)}) \geq C(d_{\Omega_{\tau(U,i),U}} - r_{\Omega_{\tau(U,i),U}\cup\{i\}}) \ . \tag{20}$$

Hence, (15) is satisfied for $\Omega = \Omega_{\tau(U,i),U}$. □

Proposition 5 has shown that $\tau(U, i)$ and the minimum slack are correctly computed by the loop at line 1. Combined with Theorem 1 this justifies the use of $minSL - e_i < 0$ on line 1 to check (EF), where the condition (EF1) is also checked. Thus, for every task i Algorithm 1 correctly detects the sets $\Omega \subseteq T \setminus \{i\}$ for which rules (EF) and (EF1) demonstrate $\Omega \ll i$.

A complete edge-finder would always choose the set Θ for each task i that yielded the strongest update to the bound of i. In the following theorem, we demonstrate that our algorithm has the slightly weaker property of soundness; that is, the algorithm updates the bounds correctly, but might not always make the strongest adjustment to a bound on the first iteration.

Theorem 2. *For every task $i \in T$, and given the strongest lower bound LB_i as specified in Definition 3, Algorithm 1 computes some lower bound LB'_i, such that $r_i < LB'_i \leq LB_i$ if $r_i < LB_i$, and $LB'_i = r_i$ if $r_i = LB_i$.*

Proof. Let $i \in T$ be any task. LB'_i is initialized to r_i. Because the value LB'_i is only updated by $\max(Dupd_i, SLupd_i, LB'_i)$ (line 1) after each detection, it follows that $LB'_i \geq r_i$. If the equality $LB_i = r_i$ holds, then no detection is found by Algorithm 1, and thus $LB'_i = r_i$ holds from the loop at line 1. In the rest of the proof, we assume that $r_i < LB_i$. By Propositions 2 and 3, there exist two task sets $\Theta_{l,u} \subseteq \Omega_{L,U} \subseteq T \setminus \{i\}$ such that $r_L \leq r_l < d_u \leq d_U < d_i$ and $r_L \leq r_i$, for which the following holds:

$$\alpha\left(\Omega_{L,U}, i\right) \quad \wedge \quad LB_i = r_{\Theta_{l,u}} + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta_{l,u}, c_i) \right\rceil \quad . \tag{21}$$

As demonstrated by Proposition 5 and Theorem 1, Algorithm 1 correctly detects the edge-finding condition; it remains only to demonstrate the computation of update values. Since (EF) and (EF1) use the same inner maximization, the following two cases hold for both rules:

1. $\boldsymbol{r_i < r_l}$: Here we prove that the update can be made using the task intervals of maximum density. According to Definition 8, we have

$$\frac{e_{\Theta_{l,u}}}{d_u - r_l} \leq \frac{e_{\Theta_{\rho(u,i),u}}}{d_u - r_{\rho(u,i)}} \quad . \tag{22}$$

   Since $(\Omega_{L,U}, \Theta_{l,u})$ allows the update of the release date of task i, we have $\operatorname{rest}(\Theta_{l,u}, c_i) > 0$. Therefore,

$$\frac{e_{\Theta_{l,u}}}{d_{\Theta_{l,u}} - r_{\Theta_{l,u}}} > C - c_i \quad . \tag{23}$$

   By relations (22) and (23), it follows that $\operatorname{rest}(\Theta_{\rho(u,i),u}, c_i) > 0$. $r_i < r_{\rho(u,i)}$ implies $r_{\rho(u,i)} + \left\lceil \frac{1}{c_i} \operatorname{rest}\left(\Theta_{\rho(u,i),u}, c_i\right) \right\rceil > r_i$. According to Proposition 4, the value $Dupd_i = r_{\rho(u,i)} + \left\lceil \frac{1}{c_i} \operatorname{rest}\left(\Theta_{\rho(u,i),u}, c_i\right) \right\rceil > r_i$ is computed by Algorithm 1 at line 1. Therefore, after the detection condition is fulfilled at line 1, the release date of task i is updated to $LB'_i = \max(Dupd_i, SLupd_i) \geq Dupd_i > r_i$.

2. $\boldsymbol{r_l \leq r_i}$: Here we prove that the update can be made using the task intervals of minimal slack. By Definition 6, we have:

$$C(d_u - r_{\tau(u,i)}) - e_{\Theta_{\tau(u,i),u}} \leq C(d_u - r_l) - e_{l,u} \ . \tag{24}$$

Adding $-c_i(d_u - r_{\tau(u,i)}) - c_i \cdot r_{\tau(u,i)}$ to the left hand side and $-c_i(d_u - r_l) - c_i \cdot r_l$ to the right hand side of (24) we get

$$-c_i \cdot r_{\tau(u,i)} - \text{rest}(\Theta_{\tau(u,i),u}, c_i) \leq -c_i \cdot r_l - \text{rest}(\Theta_{l,u}, c_i) \ . \tag{25}$$

Therefore,

$$r_{\tau(u,i)} + \frac{1}{c_i} \text{rest}(\Theta_{\tau(u,i),u}, c_i) \geq r_l + \frac{1}{c_i} \text{rest}(\Theta_{l,u}, c_i) \tag{26}$$

and

$$r_{\tau(u,i)} + \frac{1}{c_i} \text{rest}(\Theta_{\tau(u,i),u}, c_i) > r_i \tag{27}$$

since $r_l + \frac{1}{c_i} \text{rest}(\Theta_{l,u}, c_i) > r_i$. From inequality (27), it follows that

$$\text{rest}(\Theta_{\tau(u,i),u}, c_i) > 0 \tag{28}$$

since $r_{\tau(u,i)} \leq r_i$. According to Proposition 5, the value

$$SLupd_i = r_{\tau(u,i)} + \frac{1}{c_i} \text{rest}\big(\Theta_{\tau(u,i),u}, c_i\big) > r_i \tag{29}$$

is computed by Algorithm 1 at line 1. Therefore, after the detection condition is fulfilled at line 1, the updated release date of task i satisfies $LB'_i > r_i$.

Hence, Algorithm 1 is sound. □

6 Overall Complexity

According to Theorem 2, Algorithm 1 will always make some update to r_i if an update is justified by the edge-finding rules, although possibly not always the strongest update. As there are a finite number of updating sets, Algorithm 1 must reach the same fixpoint as other correct edge-finding algorithms. This "lazy" approach has recently been used to reduce the complexity of not-first/not-last filtering for cumulative resources [6], [10]; the situation differs from the previous quadratic edge-finder [9], which missed some updates altogether. Now we demonstrate that in most cases Algorithm 1 finds the strongest update immediately; when it does not, it requires at most $n - 1$ propagations.

Theorem 3. *Let $i \in T$ be any task of an E-feasible CuSP. Let $\Theta \subseteq T \setminus \{i\}$ be a set used to perform the maximum adjustment of r_i by the edge-finding rule. Let $\rho(u,i)$ be a task as given in Definition 8, applied to $i, u \in T$ with $d_u = d_\Theta$. Then Algorithm 1 performs the strongest update of r_i in the following number of iterations:*

1. *If $r_\Theta \leq r_i$, then on the first iteration,*
2. *If $r_i < r_\Theta$ then:*
   (a) If $r_i < r_\Theta \leq r_{\rho(u,i)}$, then also on the first iteration,
   (b) If $r_i < r_{\rho(u,i)} \leq r_\Theta$, then after at most $n - 1$ iterations.

Proof. Given $i \in T$, let $\Theta \subseteq T \setminus \{i\}$ be a task set used to perform the maximum adjustment of r_i by the edge-finding rule. Let $\rho(u,i)$ be the task of Definition 8 applied to i and $u \in T$ with $d_u = d_\Theta$.

1. Assume $r_\Theta \leq r_i$. By Propositions 1 and 2, and the proof of the second item of Theorem 2, formula (26) holds. By Proposition 5, when Algorithm 1 considers u in the outer loop and i in the second inner loop, it sets
$$Dupd_i = r_{\tau(u,i)} + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta_{\tau(u,i),u}, c_i) \right\rceil \geq r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil \ . \quad (30)$$
As the adjustment value of Θ is maximal, r_i is updated to $r_\Theta + \lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \rceil$.
2. Assume $r_i < r_\Theta$. We analyze two subcases:
   (a) $\boldsymbol{r_i < r_\Theta \leq r_{\rho(u,i)}}$: According to definition of task $\rho(u,i)$, we have
$$\frac{e_\Theta}{d_\Theta - r_\Theta} \leq \frac{e_{\Theta_{\rho(u,i),u}}}{d_u - r_{\rho(u,i)}} \ . \quad (31)$$
   Removing $\frac{e_{\Theta_{\rho(u,i),u}}}{d_\Theta - r_\Theta}$ from each side of (31), we get
$$e_\Theta - e_{\Theta_{\rho(u,i),u}} \leq \frac{e_{\Theta_{\rho(u,i),u}}}{d_u - r_{\rho(u,i)}} (r_{\rho(u,i)} - r_\Theta) \ . \quad (32)$$
   As the problem is E-feasible, we have
$$\frac{e_{\Theta_{\rho(u,i),u}}}{d_u - r_{\rho(u,i)}} \leq C \ . \quad (33)$$
   Combining inequalities (32) and (33) gives
$$Cr_\Theta + e_\Theta \leq Cr_{\rho(u,i)} + e_{\Theta_{\rho(u,i),u}} \ . \quad (34)$$
   Obviously, inequality (34) is equivalent to
$$r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil \leq r_{\rho(u,i)} + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta_{\rho(u,i),u}, c_i) \right\rceil \ . \quad (35)$$
   Proposition 4 shows that when Algorithm 1 considers u in the outer loop and i in the first inner loop, it sets
$$Dupd_i = r_{\rho(u,i)} + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta_{\rho(u,i),u}, c_i) \right\rceil \geq r_\Theta + \left\lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \right\rceil \ . \quad (36)$$
   As the adjustment value of Θ is maximal, r_i is updated to $r_\Theta + \lceil \frac{1}{c_i} \operatorname{rest}(\Theta, c_i) \rceil$.
   (b) $\boldsymbol{r_i < r_{\rho(u,i)} \leq r_\Theta}$: Let $\Theta^k_{\leq i} := \{j, j \in T \wedge r_j \leq r_i \wedge d_j \leq d_\Theta\}$ and $\Theta^k_{>i} := \{j, j \in T \wedge r_j > r_i \wedge d_j \leq d_\Theta\}$ be sets of tasks defined at the k$^{\text{th}}$ iteration of Algorithm 1. If the maximum adjustment is not found after this iteration, then at least one task is moved from $\Theta^k_{>i}$ to $\Theta^k_{\leq i}$. Indeed, if at the k$^{\text{th}}$ and k+1$^{\text{th}}$ iteration, $\Theta^k_{\leq i} = \Theta^{k+1}_{\leq i}$, $\Theta^k_{>i} = \Theta^{k+1}_{>i}$ and the maximum adjustment is not found, then the tasks $\tau(u,i) \in \Theta^k_{\leq i} = \Theta^{k+1}_{\leq i}$ (Definition 6) and $\rho(u,i) \in \Theta^k_{>i} = \Theta^{k+1}_{>i}$ (Definition 8) are the same for both iterations. Therefore, at the k+1$^{\text{th}}$ iteration, no new adjustment

is found, yet the maximum adjustment of the release date of task i is not reached, thus contradicting the soundness of Algorithm 1. Hence, the maximum adjustment of the release date of task i is reached after at most $|\Theta^1_{>i}| \leq n-1$ iterations. □

We argue that, in practice, the possibility of our algorithm using multiple propagations to find the strongest bound is not significant. In the first place, edge-finders are not idempotent; adjustment to the release dates and deadlines of the tasks is not taken into account during one iteration, so additional propagations are always required to reach a recognizable fixpoint. Furthermore, in actual cumulative problems, there are typically a relatively small number of task sets that could be used to update the start time of a given task, so the number of propagations should not normally approach the worst case. This claim is borne out by the experimental observations reported in the next section.

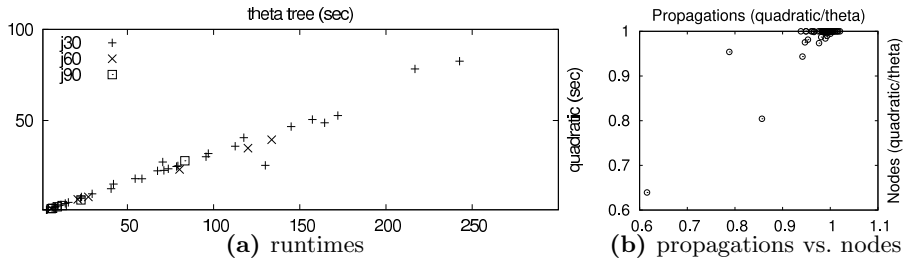
7 Experimental Results

The new edge-finding algorithm presented in section 5 was implemented in C++ using the Gecode 3.4.2 [5] constraint solver. The Gecode cumulative propagator for tasks of fixed duration is a sequence of three filters: the $\mathcal{O}(kn \log n)$ edge-finding algorithm from [13], overload checking, and time tabling. We tested this propagator against a modified version that substituted the new quadratic edge-finding filter for the Θ-tree filter.

Tests were performed on the single-mode J30, J60, and J90 test sets of the well-established benchmark library PSPLib [7]. Each data set consists of 480 instances, of 30, 60, and 90 tasks respectively; these tasks require multiple shared resources, each of which was modeled with a cumulative constraint. Precedence relations were enforced as a series of linear relations between task start and end times. Branch and bound search was used to find the minimum end time for the project; variables were selected by minimum domain size, with ties broken by selecting the variable occuring in the most propagators, while values were selected starting with the minimum. Tests were performed on a 3.07 GHz Intel Core i7 processor with a time limit of 300 seconds; only tests for which both propagators were able to find the best solution within 300 seconds are included in Fig. 3 (8 instances in which only the quadratic propagator was able to find a solution in the time available were discarded)[1]. Each instance was run three times, with the best result for each filtering algorithm reported.

Our tests showed that the quadratic edge-finder was faster in almost all test instances, with a proportional speedup increasing with the size of the instance. Of the 1034 instances solved by both filters, only four instances from the j3036 group and the j601_1 instance were solved more quickly by the Θ-tree filter. Figure 3a compares the runtimes of the hardest instances (with runtime greater than 1 second on the Θ-tree filter).

[1] Detailed experimental results: `http://user.it.uu.se/~jossc163/quadef2011`.



(a) runtimes **(b)** propagations vs. nodes

Fig. 3. Two comparisons of Θ-tree vs. quadratic edge-finding: (a) runtimes for instances where both methods found the best solution, and (b) the proportion of quadratic to Θ-tree propagation counts, and node counts.

In order to determine the difference in propagation strength between the two filters, we instrumented the two propagators to count the number of executions of the cumulative propagator in each of the 1034 solved instances. As expected, the number of propagations was different in 122 of these instances; however, only in 35 of those instances was the number of propagations required by the quadratic filter greater. While this could suggest that the quadratic filter reaches fixpoint more quickly than the Θ-tree filter in some cases, a more likely explanation is that the domain reductions made by the two filters before reaching fixpoint were different enough to affect the filtering strength of the other propagators used in the problem. Figure 3b compares the number of propagations (shown as the proportion of quadratic propagations to Θ-tree propagations) to the number of nodes in the search tree for each algorithm. We observe that, even in those instances where the quadratic filter required a larger number of propagations, the number of nodes in the search tree of the quadratic algorithm was always less than equal to those in the Θ-tree filter search tree, implying that the quadratic algorithm reaches at least as strong a fixpoint as the Θ-tree filter.

8 Conclusion

In this paper, we have presented a quadratic edge-finding filtering rule for cumulative scheduling that reaches the same fixpoint as previous algorithms, possibly after more propagations. While its complexity does not strictly dominate that of Vilím's $\mathcal{O}(kn \log n)$ algorithm, experimental results demonstrate that on a standard benchmark suite our algorithm is substantially faster. Future work will focus on finding a complete quadratic edge-finder, a similar algorithm for extended edge-finding, and investigating the use of Θ-trees to increase the efficiency of finding maximum density.

Acknowledgements. The third author is supported by grant 2009-4384 of the Swedish Research Council (VR). The authors would also like to thank Christian Schulte for his assistance with the Gecode cumulative propagator.

References

1. Baptiste, P., Le Pape, C.: Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. In: Smolka, G. (ed.) CP 1997. LNCS, vol. 1330. Springer, Heidelberg (1997)
2. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-based scheduling: applying constraint programming to scheduling problems. Kluwer, Boston (2001)
3. Carlier, J., Pinson, E.: Adjustment of heads and tails for the job-shop problem. European Journal of Operational Research 78, 146–161 (1994)
4. Caseau, Y., Laburthe, F.: Improved CLP scheduling with task intervals. In: Van Hentenryck, P. (ed.) ICLP94, pp. 369–383. MIT Press, Boston (1994)
5. Gecode Team: Gecode, a generic constraint development environment (2006), `http://www.gecode.org`
6. Kameugne, R., Fotso, L.P.: A not-first/not-last algorithm for cumulative resource in $\mathcal{O}(n^2 \log n)$ (2010) (accepted to CP 2010 Doctoral Program)
7. Kolisch, R., Sprecher, A.: PSPLIB – A project scheduling problem library. European Journal of Operational Research 96(1), 205–216 (1997)
8. Mercier, L., Van Hentenryck, P.: Edge finding for cumulative scheduling. INFORMS Journal on Computing 20(1), 143–153 (2008)
9. Nuijten, W.: Time and resource constrained scheduling: a constraint satisfaction approach. PhD thesis, Eindhoven University of Technology (1994)
10. Schutt, A., Wolf, A.: A New $\mathcal{O}(n^2 \log n)$ Not-First/Not-Last Pruning Algorithm for Cumulative Resource Constraints. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 445–459. Springer, Heidelberg (2010)
11. Vilím, P.: Global constraints in scheduling. PhD thesis, Charles University, Prague (2007)
12. Vilím, P.: Max energy filtering algorithm for discrete cumulative resources. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 66–80. Springer, Heidelberg (2009)
13. Vilím, P.: Edge Finding Filtering Algorithm for Discrete Cumulative Resources in $\mathcal{O}(kn \log n)$. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 802–816. Springer, Heidelberg (2009)
14. Wolf, A., Schrader, G.: $\mathcal{O}(n \log n)$ overload checking for the cumulative constraint and its application. In: Umeda, M., Wolf, A., Bartenstein, O., Geske, U., Seipel, D., Takata, O., et al. (eds.) INAP 2005. LNCS (LNAI), vol. 4369, pp. 88–101. Springer, Heidelberg (2006)